algorithms

MDPI

Article

Indetermsoft-Set-Based D* Extra Lite Framework for Resource Provisioning in Cloud Computing

Bhargavi Krishnamurthy [1,*] and Sajjan G. Shiva [2,*]

1 Department of CSE, Siddaganga Institute of Technology, Tumakuru 572103, Karnataka, India
2 Department of CS, University of Memphis, Memphis, TN 38152, USA
* Correspondence: bhargavik@sit.ac.in (B.K.); sshiva@memphis.edu (S.G.S.)

Abstract: Cloud computing is an immensely complex, huge-scale, and highly diverse computing platform that allows the deployment of highly resource-constrained scientific and personal applications. Resource provisioning in cloud computing is difficult because of the uncertainty associated with it in terms of dynamic elasticity, rapid performance change, large-scale virtualization, loosely coupled applications, the elastic escalation of user demands, etc. Hence, there is a need to develop an intelligent framework that allows effective resource provisioning under uncertainties. The Indetermsoft set is a promising mathematical model that is an extension of the traditional soft set that is designed to handle uncertain forms of data. The D* extra lite algorithm is a dynamic heuristic algorithm that makes use of the history of knowledge from past search experience to arrive at decisions. In this paper, the D* extra lite algorithm is enabled with the Indetermsoft set to perform proficient resource provisioning under uncertainty. The experimental results show that the performance of the proposed algorithm is found to be promising in performance metrics such as power consumption, resource utilization, total execution time, and learning rate. The expected value analysis also validated the experimental results obtained.

Keywords: resource provisioning; uncertainty; D* extra lite; cloud computing; Indetermsoft set; cloud computing

check for updates

Citation: Krishnamurthy, B.; Shiva, S.G. Indetermsoft-Set-Based D* Extra Lite Framework for Resource Provisioning in Cloud Computing. *Algorithms* **2024**, *17*, 479. https://doi.org/10.3390/a17110479

Academic Editors: Frank Werner and Claudio Savaglio

Received: 11 June 2024
Revised: 29 August 2024
Accepted: 16 October 2024
Published: 25 October 2024

1. Introduction

Cloud computing is a highly expansive platform that supports a broad spectrum of applications, catering to diverse needs across various domains. Its vast infrastructure allows for the scalable deployment of both resource-intensive scientific simulations and lightweight personal applications. By offering on-demand access to computing resources, cloud computing enables flexibility and efficiency, accommodating everything from large-scale data processing to everyday tasks. This adaptability makes it an invaluable tool for organizations and individuals alike, providing the necessary resources to handle complex workloads and drive innovation across different fields. Effective resource management is composed of several activities, which include the provisioning of resources, the reporting of provisioning, task scheduling, and thermal management. The resource management system must be capable of dealing with the huge size, heterogeneity, and changing workload demands of cloud users. The main aim of the resource-provisioning scheme is to achieve the maximum data transfer rate with the minimum incurred cost of transfer. For applications utilizing the cloud, their Quality of Service (QoS) must be upgraded without violating the Service-Level Agreement (SLA). While the resources are provisioned, resource availability must be ensured, deployment must be optimized, and interdependency between the user tasks must be managed very well. The improper provisioning of resources leads to an increase in the host machine downtime, compromised service, the inefficient functioning of the application, energy wastage, prolonged time to scale up/down, inability to satisfy the cost goal of users, etc. [1–3].

Modern cloud computing structures are utilized to serve computation-intensive applications with large datasets. Resource provisioning is an important aspect of accommodating this use. Potential sources that cause interruption to effective resource provisioning are uncertain operating conditions, performance fluctuations, the improper positioning of the cloud, and the complex infrastructure of a multi-cloud environment. Several forms of uncertainty in the cloud include dynamic elasticity, rapid performance change, large-scale virtualization, loosely coupled applications, the elastic escalation of user demands, etc. These uncertainties make resource provisioning a difficult activity. It is also impossible to determine the workload demands and to obtain the exact knowledge of system parameters like processor speed and bandwidth. Hence, there is a need to develop an intelligent framework with a self-learning ability to make resource-provisioning decisions by properly handling these uncertainties [4,5].

The classical soft set deals only with the determinate form of data, whose values are obviously certain and precise. However, in real-world scenarios, there are several sources that generate uncertainty, which include a lack of information and ignorance. The Indetermsoft set is an extension of the traditional soft set that is designed to handle the uncertain form of data. The word "indeterm" represents indeterminate, reflecting the conflicting and unique form of output. The mathematical definition of the Indetermsoft set is as follows: Let U be the universe of discourse, H be the non-empty subset of U, and P(H) be the power set of H. The Indetermsoft set function $ISSF : A \to P(H)$ satisfies three properties: the set A has indeterminacy, P(H) has some indeterminacy, and there exists an attribute value $v \in A$ such that F(v) is indeterminate [6–8].

D* lite is exactly the reverse of the A* algorithm and is much simpler. The trivial A* algorithm is executed in reverse order; i.e., it begins from the goal state and traverses to the start state. It first determines the current solution and goes into the waiting state until an obstacle occurs. Then, D* lite performs re-planning and incrementally repairs the path by keeping the modifications around the robot's current pose [9,10]. D* extra lite is a novel and general-purpose algorithm that performs a shortest-path search using an incremental search approach. It performs a fast re-initialization of the search space to determine the shortest path in an unknown or highly dynamic environment. In this paper, the novel D* extra lite algorithm is enriched with the Indetermsoft set mathematical model, which performs a superior re-initialization process under cloud uncertainty to provide stable resource-provisioning decisions.

The objectives of this paper are as follows:

- Providing a brief introduction to the need for effective resource provisioning in uncertain cloud computing systems;
- The efficient handling of parameter uncertainty in the user tasks and virtual machines using the Indetermsoft set mathematical model;
- The design of a novel Indetermsoft-set-based D* extra lite framework for resource provisioning in the cloud;
- An experimental evaluation of the D* extra lite framework performance using the Google Cluster dataset and the Bitbrains dataset using the CloudSim 3.0 open-source framework;
- An expected value analysis and validation of the D* extra lite framework in a dynamic cloud scenario with respect to future time intervals.

The remaining sections of this paper are organized as follows. Section 2 discusses related work. Section 3 provides a mathematical definition of the system model, along with the performance objectives. Section 4 presents the novel Indetermsoft-set-based D* extra lite framework. Section 5 provides the mathematical modeling of the performance objectives that were considered for evaluation. Section 6 deals with the results and discussion by considering the Google Cluster dataset and the Bitbrains dataset. Finally, Section 7 presents the conclusions.

2. Related Work

This section provides a comprehensive overview of the existing literature and developments pertinent to resource provisioning in cloud computing. Through this review of previous research, methodologies, and technological advancements, we establish the foundation/scope for the proposed work.

Shreshth et al. [11] present an artificial-intelligence-based holistic approach named HUNTER for resource management in cloud computing. They view optimizing energy emission in cloud data centers as a multiple-objective-based scheduling problem. A large amount of energy is consumed by cloud data centers, and hence, there is a need to optimize energy emissions. One of the key factors to consider while optimizing energy consumption is reducing the number of thermal hotspots which lead to the degradation of system performance. Three important models are considered for resource management: cooling, thermal, and energy. A gated graph convolutional network is viewed as a surrogate model for optimizing Quality of Service (QoS) to perform optimal task scheduling. The continuous training of the model helps in quick adaption to dynamic scenarios by providing the permission to change the scheduling decisions through task migration. Power performance is used as a heuristic to efficiently balance the load among the cloud hosts. The performance of the HUNTER is evaluated through the CloudSim simulation toolkit and is good with respect to energy conservation. However, the approach exhibits poor scalability as it cannot scale up to large-scale graphs with higher node degrees.

Spyridon et al. propose an auto-scaling framework for resource provisioning in a cloud computing environment [12]. The over- and under-provisioning of resources results in a loss of revenue for the cloud brokers whose primary function is to select, manage, and provide the resources in a complex heterogeneous environment. Since the client resource requirements are uncertain, it becomes difficult for the cloud broker to predict and process the client resource demands. Here, the resource-provisioning problem is divided into two stages: resource selection and resource management. The resource selection problem deals with the process of selecting the services that meet the requirements of multiple cloud service providers. The resource management problem deals with the effective maintenance of cloud resources in terms of resource utilization and overhead maintenance. Both selection and management of resources have been considered as decision-making problems that focus on matching the resource requests with services provided. The cloud users make use of virtualized resources in order to benefit from long-term pricing strategies. For providing cost-effective solutions, a precise estimation of the upcoming workload is required. An adaptive auto-scaling framework for resource provisioning that uses historical time-series data for training a K-means-enabled CNN framework to categorize the future workload demands as low, medium, or high as per their CPU utilization rate is proposed here. From the performance evaluation, it is observed that the solution deployment cost is minimal. However, the framework exhibits higher sensitivity towards initial parameter setting and an inability to handle categorical data in large-state-space environments.

Kumar et al. present an efficient algorithm for resource provisioning for the efficient execution of workflows in cloud computing platforms [13]. The workflow considered here is composed of tasks exhibiting varying resource requirements in terms of memory storage, memory type, and computation speed. Improper mapping of the workflows to resources leads to wastage of resources and increased makespan time. Here, the workflow is divided into three categories: compute-intensive, memory-intensive, and storage-intensive. The proposed algorithm operates in two phases to provision resources precisely by distinguishing the tasks as computation-intensive and non-computation-intensive. The workflow model is composed of limited information about the task contained in it, which makes it applicable to real-time scenarios. The Amazon EC2 cloud model is considered for offering on-demand computational resources for applications using the presented algorithm. However, the approach is found to be static and applies a standard set of operations to process computation-intensive and non-computation-intensive tasks. This limits the practical applicability of the approach.

Mao et al. discuss a game approach based on mean-field theory for resource management in cloud computing [14]. Resource management is viewed as one of the prominent problems in serverless cloud computing environments. Multiple users compete for resources which usually suffer from scalability issues. Here, an actor–critic learning algorithm is developed to effectively deal with large-state-space environments. The algorithm is implemented by considering linear and neural network approximations. The mean-field approach is compatible with several forms of function approximations. Theoretically, convergence to Nash equilibrium is achieved under linear and softmax approximations. The performance of this approach is better in terms of resource utilization, but the time to converge towards better resource-provisioning policies is excessive.

Lucia et al. [15] present an adaptive reinforcement-learning-based approach for resource provisioning in serverless computing. Cloud service providers expect a resource-provisioning scheme to be flexible to meet the fluctuating demands of the customers. A request-based policy is proposed here in which the resources are scaled for the maximum number of requests processed in parallel. The performance is strongly influenced by the predetermined concurrency level. The performance evaluation indicates that with a limited number of iterations, the model formulates efficient scaling policies. But identifying the concurrency level that provides the maximum QoS is difficult because of the varying workload, complex infrastructure, and high latency.

Sangeetha et al. discuss a novel resource management framework based on deep learning [16]. Increased multimedia traffic in the cloud leads to minimum extensibility of a service portfolio and poor resource management. A gray-wolf-optimization-based resource allocation strategy that mimics the hunting behavior of grey wolves is proposed here. The deep neural network utilized here provides routing direction based on the data input rate and storage availability. The neural network operates in two phases: data pre-processing and routing, and controlling application. While the delay in processing the requests is reduced by this policy, it suffers from a poor search ability and a slow convergence rate.

Saxena et al. develop an elastic resource management framework to provide cloud services with high availability [17]. There is a very high demand for resources on the cloud, and the failure to provide on-demand services leads to load imbalance, performance degradation, and excessive power consumption. An online failure predictor that predicts the possibility of the virtual machines resulting in resource starvation due to a resource contention situation is developed. The server under operation is continuously monitored with the aid of a power analyzer, resource monitor, and thermal analyzer. The virtual machines that exhibit a high probability of failure are assigned to the fault tolerance unit that can handle all outages and performance degradation. The virtual machine failure prediction accuracy in this technique is poor, leading to poor resource management policies.

In summary, the existing works exhibit the following drawbacks:

- Inability to determine the parameter uncertainty in the user tasks and virtual machines.
- Conventional resource-provisioning approaches are static in nature, limiting their practical application.
- Rule-based approaches are time-consuming and hard to scale, and the rate of virtual machine violations in terms of cost and response time is very high.
- Most of the heuristic approaches exhibit a higher tendency for premature convergence under uncertainty.
- Predictive approaches exhibit poor prediction accuracy leading to over- or under-utilization of resources.
- The computational complexity of the soft computing approaches is high as they deal with a large number of optimization parameters.
- The learning algorithms fail to consider the highly dynamic operating conditions of a cloud system. As a result, they cannot handle the dynamic task scheduling and dynamic placement of resources efficiently.

3. System Model

This section provides the details of the structure, components, and interactions among the components of the proposed cloud resource-provisioning system. A detailed representation of its operational dynamics and a discussion of its theoretical foundations are also presented. The Indetermsoft-set-based D* extra lite framework for edge computing systems is composed of three functional modules: Indetermsoft set task manager, Indetermsoft set resource manager, and D* extra lite.

The user submits a set of tasks (UTs) to the system accessibility layer, i.e., $UT_1 = \{ut_1, ut_2, ut_3, \ldots, ut_m\}$, $UT_2 = \{ut_1, ut_2, ut_3, \ldots, ut_m\}$, $\ldots, UT_m = \{ut_1, ut_2, ut_3, \ldots, ut_m\}$. The system accessibility layer places these user tasks into the task queue of the task manager. The Indetermsoft set task manager applies the Indetermsoft set function (ISF) to the user tasks. That is,

$$\begin{gathered} ISF(UT_1) = \{ isf(ut_1), isf(ut_2), isf(ut_3), \ldots, isf(ut_m) \}, \\ ISF(UT_2) = \{ isf(ut_1), isf(ut_2), isf(ut_3), \ldots, isf(ut_m) \}, \\ ISF(UT_m) = \{ isf(ut_1), isf(ut_2), isf(ut_3), \ldots, isf(ut_m) \} \end{gathered} \tag{1}$$

Similarly, the resource center is composed of several resource instances, each instance consists of a set of hosts, and a set of virtual machines are mounted on each host.

$$\begin{gathered} RCI_1 = \{H_1(vm_1,\ vm_2, \ldots, vm_n), \ldots, H_n(vm_1,\ vm_2, \ldots, vm_n)\ \}, \\ RCI_2 = \{H_1(vm_1,\ vm_2, \ldots, vm_n), \ldots, H_n(vm_1,\ vm_2, \ldots, vm_n)\ \}, \\ RCI_n = \{H_1(vm_1,\ vm_2, \ldots, vm_n), \ldots, H_n(vm_1,\ vm_2, \ldots, vm_n)\ \} \end{gathered} \tag{2}$$

The Indetermsoft set resource manager applies the Indetermsoft set function to the resource instances.

$$\begin{gathered} ISF(RCI_1) = \{ isf(H_1(vm_1,\ vm_2, \ldots, vm_n)) \ldots isf(H_n(vm_1,\ vm_2, \ldots, vm_n)) \}, \\ ISF(RCI_2) = \{ isf(H_1(vm_1,\ vm_2, \ldots, vm_n)) \ldots isf(H_n(vm_1,\ Vm_2, \ldots, vm_n)) \}, \\ ISF(RCI_n) = \{ isf(H_1(vm_1,\ vm_2, \ldots, vm_n)) \ldots isf(H_n(vm_1,\ vm_2, \ldots, vm_n))\ \} \end{gathered} \tag{3}$$

The D* extra lite functional module combines the Indetermsoft set function of user tasks and the resource. It is an incremental heuristic search algorithm which is a dynamic form of the A* algorithm that generates D* extra lite task resource-provisioning policies:

$$D^*(RPP) = \{\ D^*(RPP_1),\ D^*(RPP_2), \ldots, D^*(RPP_n)\ \} \tag{4}$$

The following performance objectives (POs) are set for the D* extra lite framework.

PO1: Power Consumption (PC(D* extra lite)): The power consumption of the D* extra lite framework is the summation of the power consumption of the resource instances $PC(RCI_i)$.

$$PC(D^*\ extra\ lite) = \sum_{i=1}^{i=n} PC(RCI_i) \tag{5}$$

where $PC(RCI_i)$ is determined by the summation of the power consumption of the host $PC(H_i)$, i.e., $PC(RCI_i) = \sum_{i=1}^{i=n} PC(H_i)$. Here, $PC(H_i)$ is determined by considering the maximum power consumption state $PC(vm_i^{max})$, minimum power consumption state $PC(vm_i^{min})$, and idle state power consumption $PC\left(vm_i^{idle}\right)$ of the virtual machine:

$$PC(H_i) = \left[\ PC(vm_i^{max}) - PC\left(vm_i^{min}\right)\right] * RU(vm_i) + PC\left(vm_i^{idle}\right)\Big] \tag{5a}$$

PO2: Resource Utilization (RU(D* extra lite)): The resource utilization of the D* extra lite framework is computed by the summation of the resources utilized by the resource instances $RU(RCI_i)$.

$$RU(D^* extra\ lite) = \sum_{i=1}^{i=n} RU(RCI_i) \tag{6}$$

where $RU(RCI_i)$ is determined by the summation of the resources consumed by the host $RU(H_i)$, i.e., $RU(RCI_i) = \sum_{i=1}^{i=n} RU(H_i)$. Here, $RU(H_i)$ is the measure of the utilities of the virtual machines that are used by the user tasks. It is determined by computing the over-utilized resource center instances RCI_i^{over} and under-utilized resource center instances RCI_i^{under} among the set of available resource center instances N_{RCI}.

$$RU(D^*\ extra\ lite) = \sum_{i=1}^{i=n} \left(RCI_i^{over} - RCI_i^{under} \right) / N_{RCI} \tag{6a}$$

PO3: Total Execution Time (TET(D* extra lite)): The total execution time of the D* extra lite framework is the time taken to assign the user tasks to virtual machines. It is the summation of total execution time of the Indetermsoft set function of user tasks ISF(UT_i), Indetermsoft set function of resource instances $ISF(RCI_i)$, and D* policy $D^*(RPP_i)$.

$$\begin{gathered} TET(D^*\ extra\ lite) = \\ \sum_{i=1}^{i=m} TET(ISF(UT_i)) + \sum_{i=1}^{i=n} TET(ISF(RCI_i)) + \sum_{i=1}^{i=n} TET(D^*(RPP_i)) \end{gathered} \tag{7}$$

PO4: Learning Rate (LR$\left(\text{D}^*\right.$ extra lite)): The learning rate of the D* extra lite framework is the speed at which the Indetermsoft set function of user tasks $ISF(UT_i)$ is mapped to the Indetermsoft set function of resource instances $ISF(RCI_i)$. It is computed by considering the total execution time for the formulation of the first resource-provisioning policy $ET(D^*(RPP_1))$ and the number of resource-provisioning policies formulated $N(D^*(RPPP))$.

$$LR(D^*\ extra\ lite) = TET(D^*(RPP_1)) * N(D^*(RPP)) \tag{8}$$

4. Proposed Work

This section outlines the innovative contributions and research directions that form the core of this research study and presents the novel ideas, methodologies, and proposed solutions, emphasizing their significance and potential impact on the resource-provisioning field.

As shown in Figure 1, the Indetermsoft-set-based D* extra lite framework is composed of three distinct modules. The users submit requests to the system accessibility layer. The task manager is responsible for monitoring the resources and tasks. The resource center is composed of a set of virtual machines that are hosted on several physical machines. The uncertainty in the incoming tasks is managed by the Indetermsoft set task manager. Similarly, the uncertainty in the resources is managed by the Indetermsoft set resource manager. The D* extra lite algorithm is executed on the Indetermsoft set of tasks and resources. D* extra lite is also referred to as dynamic A*, which determines an ideal path between the starting point and goal of an application. The obstacles that occur are handled efficiently when they are encountered on the path towards the destination.

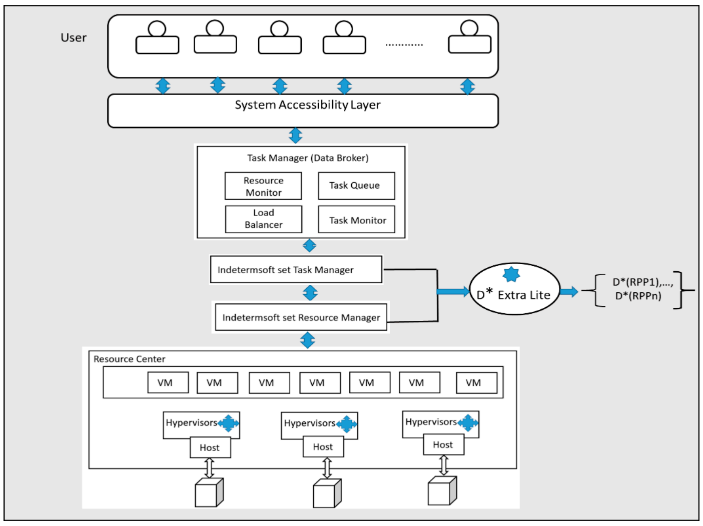


Figure 1. Indetermsoft-set-based D* extra lite framework.

4.1. *Indetermsoft Set Task Manager (ISSTM)*

The ISSTM component of the D* extra lite framework accepts a set of user tasks UT as inputs; $\mathrm{UT} = \{\mathrm{UT}_1, \mathrm{UT}_2, \mathrm{UT}_3, \ldots, \mathrm{UT}_\mathrm{m}\}$. It applies the Indetermsoft set function to generate the Indetermsoft set function ISF of the user task set. $ISF(UT) = \{ISF(UT_1), ISF(UT_2), ISF(UT_1), \ldots, ISF(UT_m)\}$. The Indetermsoft set function handles the indeterminate and conflicting parameters of the user tasks by mapping the attributes of user tasks to the power set of the user task set. The working of the ISSTM module is depicted in Algorithm 1, which consists of a training phase and a testing phase. During the training phase, for each set of user tasks, the Indetermsoft set function is applied to generate the Indetermsoft set function of the user task set. Likewise, during the testing phase, the cumulative aggregation of the Indetermsoft set function of the user task set is computed.

Algorithm 1: Working of ISSTM

```
1: Start
2: Input user task set
        $\mathrm{UT} = \{\mathrm{UT}_1, \mathrm{UT}_2, \mathrm{UT}_3, \ldots, \mathrm{UT}_\mathrm{m}\}$
3: Output ISF of user task set
        $ISF(UT) = \{ISF(UT_1), ISF(UT_2), ISF(UT_1), \ldots, ISF(UT_m)\}$
4: Training phase of ISSTM
5 :  For each training user task set $UT_\mathrm{i} \in UT$ do
6:     For each training user task set attributes $UT_i^{at} = \{ut_1^{at}, ut_2^{at}, \ldots, ut_m^{at}\}$ in UT do
7:          Train $\forall\ isf_\mathrm{i}(ut_k)^\mathrm{at} \in \mathrm{ut}^\mathrm{at}$ initialize $\sigma\left(isf_\mathrm{i}(ut_k)^\mathrm{at}\right) = \mathrm{NULL}$
8:       Calculate training ISF of user tasks
9:              $ISF(UT_\mathrm{i}) : ut_k^{at} \to P\left(H\left(UT_i^{at}\right)\right), H \subseteq UT$
10:    End For
11: End For
12: Testing phase of ISSTM
13 :     For each testing user task set $UT_\mathrm{i} \in UT$ do
14 :       For each testing user task set attributes $UT_i^{at} = \{ut_1^{at}, ut_2^{at}, \ldots, ut_m^{at}\}$ in UT do
15 :         Test $\forall\ isf_\mathrm{i}(ut_k)^\mathrm{at} \in \mathrm{ut}^\mathrm{at}$ initialize $\sigma\left(isf_\mathrm{i}(ut_k)^\mathrm{at}\right) = \mathrm{NULL}$
16:           Compute aggregation of Indetermsoft set function
                $ISF(UT) ::= ISF(UT_i) \cup \left(UT_i^{at}, isf_\mathrm{i}(ut_k)^\mathrm{at}\right)$
17:    End For
18: End For
19 :  Output $ISF(UT) = \{ISF(UT_1), ISF(UT_2), ISF(UT_1), \ldots, ISF(UT_m)\}$
20: Stop
```

4.2. Indetermsoft Set Resource Manager (ISSRM)

The ISSRM component of the D* extra lite framework receives the set of resource center instances RCI as inputs. $RCI = \{RCI_1, RCI_2, RCI_3, \ldots, RCI_m\}$. It applies the Indetermsoft set function to generate the Indetermsoft set function ISF of the resource center instance set. $ISF(RCI) = \{ISF(RCI_1), ISF(RCI_2), ISF(RCI_3), \ldots, ISF(RCI_m)\}$. The Indetermsoft set function handles the indeterminate, conflicting parameters of the resource center instances by mapping the attributes of resource center instances to the power set of the resource center instance set. The working of the ISSRM module is shown in Algorithm 2, which consists of training and testing phases. During the training phase, for every set of resource center instances, the Indetermsoft set function is applied to generate the Indetermsoft set function of resource center instances. Likewise, during the testing phase, the cumulative aggregation of the Indetermsoft set function of the resource center instance set is computed.

Algorithm 2: Working of ISSRM

```
1: Start
2: Input user task set
        $\mathrm{RCI} = \{\mathrm{RCI}_1, \mathrm{RCI}_2, \mathrm{RCI}_3, \ldots, \mathrm{RCI}_\mathrm{m}\}$
3: Output ISF of user task set
        $ISF(RCI) = \{ISF(RCI_1), ISF(RCI_2), ISF(RCI_3), \ldots, ISF(RCI_n)\}$
4: Training phase of ISSRM
5 : For each training resource center instance $RCI_\mathrm{i} \in RCI$ do
6:      For each training resource center instance attributes do
            $RCI_i^{at} = \{H_i(vm_1^{at}), \ldots, H_i(vm_i^{at})\}$ in $RCI$
7:          Train $\forall isf_\mathrm{i}(H_i(vm_1^{at}) \in H_i(vm)^{\mathrm{at}})$ initialize $\sigma(isf_i(H_i(vm_k^{at}))) = \mathrm{NULL}$
8:      Calculate training ISF of resource center instances
9:          $ISF(RCI_\mathrm{i}) : H_i(vm_k^{at}) \to P(H(H_i(vm_k^{at}))), H \subseteq RCI$
10:   End For
11: End For
12: Testing phase of ISSRM
13 :   For each testing user task set $RCI_\mathrm{i} \in RCI$ do
14:         For each testing user task set attributes do
            $RCI_i^{at} = \{H_i(vm_1^{at}), H_i(vm_i^{at})\}$ in $RCI$
15:         Test $\forall isf_\mathrm{i}\left(H_i(vm_1^{at}) \in H_i(vm)^{\mathrm{at}}\right)$ initialize $\sigma(isf_i(H_i(vm_k^{at}))) = \mathrm{NULL}$
16:         Compute aggregation of Indetermsoft set function
            $ISF(RCI) ::= ISF(RCI_i) \cup (RCI_i^{at}, \forall\forall isf_\mathrm{i}(H_i(vm_k^{at})))$
17:   End For
18: End For
19 : Output $ISF(RCI) = \{ISF(RCI_1), ISF(RCI_2), ISF(RCI_3), \ldots, ISF(RCI_n)\}$
20: Stop
```

4.3. D* Extra Lite (D*EL)

The D*EL component of the D* extra lite framework accepts the Indetermsoft set function of the user task set $ISF(UT) = \{ISF(UT_1), ISF(UT_2), \ldots, ISF(UT_m)\}$ and Indetermsoft set function of resource center instances to generate D* extra lite resource-provisioning policies $D^*(RPP) = \{ D^*(RPP_1), \ldots, D^*(RPP_n) \}$. The working of the D*EL component is shown in Algorithm 3, which is composed of training and testing phases. The user tasks and resource center instances are represented in terms of an acyclic tree composed of nodes and edges. During training, D*EL uses an incremental heuristic search technique for the robust navigation of user tasks to ideal resource center instances. During testing, for the incoming user tasks, an ideal path is determined by using a priority queue data structure. The time incurred in training and testing is less as the user tasks are aligned in a priority queue which prevents frequent reordering and fast re-planning during user task navigation. The efficiency lies in re-expanding the parts of the search space that are registered for changes. The CALCULATE KEY function uses the key value for assigning priority to the list elements, and this value is determined through the sum of heuristic values. The REINITIALIZE function performs fast re-initialization of the affected search space using a search tree. Quick re-computation of the optimal path is performed by keeping track of visited and unvisited states of the resource center instances using the CUT BRANCHES function.

Algorithm 3: Working of D*EL

```
1: Start
2: Input
        $ISF(UT) = \{ISF(UT_1), ISF(UT_2), ISF(UT_1), ..., ISF(UT_m)\}$
                $ISF(RCI) = \{ISF(RCI_1), ISF(RCI_2), ISF(RCI_3), ..., ISF(RCI_n)\}$
3 : Output $D^*(RPP) = \{D^*(RPP_1), D^*(RPP_2), \ldots, D^*(RPP_n)\}$
4: Function CALCULATE KEY (State S)
5 :         Return $[g(s) + h(S_{start}, S) + K_m; g(s)]$, where h is the heuristic value,
                $g(s)$ is the cos t from goal state, and $K_m$ is the bias value.
6: Function SOLUTION FOUND ()
7 :   Return TOP OPEN $= S_{start}$ or Visited $(S_{start})$ AND NOT OPEN $(S_{start})$
8: Function INITIALIZE ()
9 :    $K_m = 0$, Visited $(S_{goal})$ = True, Parent $(S_{goal})$ = NULL, g $(S_{goal}) = 0$
10 :   PUSH OPEN $(S_{goal}$, CALCULATE KEY $(S_{goal}$))
11: Function SEARCH STEP ()
12:   s = TOP OPEN ()
13:   POP OPEN ()
14 :    $k_{old} = key(s)$
15 :      $k_{new}$ = CALCULATE KEY $(S_{goal})$
16 :      if $k_{old} < k_{new}$ then
17:      PUSH OPEN(s, CALCULATE KEY(s))
18:      else
19 :          for all $s' \in Pred(s)$ do
20 :      if NOT VISITED $(s')$ OR $g(s') > cost(s', s) + g(s)$then
21 :                  $Parent(s') = s$
22 :          $g(s') = cost(s', s) + g(s)$
23 :          if NOT VISTED $(s')$ then
24 :                              VISITED $(s')$ = true
25 :            PUSH OPEN $(s'$, CALCULATE KEY $(s')$)
26:       End for
27:  Function REINITIALIZE ()
28:    if any edge cost changed then
29:       CUT BRANCHES
30 :        if $seeds \neq \varnothing$ then
31 :          $K_m = K_m + h(S_{last}, S_{start})$
32 :     $S_{last} = S_{start}$
33 :     for all s $\in Seeds\ do$
34 :              if $visited(s)$ AND NOT OPEN$(S)$ THEN
35:        PUSH OPEN (s, CALCULATE KEY(s))
36 :     seed $= \varnothing$
37:    Function CUT BRANCHES ()
38:        Reopen_start = false
39:        For all directed edges (u, v) with changed cost do
40:     if visited (u) AND visited (v) then
41:        cold = cost (u, v)
42:        Update edge cost (u, v)
43:        if cold > cost (u, v) then
44 :                if $g(start) > g(v) + cost(u, v) + h(S_{start}, u)$ then
45:          reopen_start = true
46 :           seeds = seeds $\cup v$
47:           else if cold < cost (u, v) then
48 :               if $parent\ (u) = v$ then
49:        CUT BRANCHES (u)
50 :         if $reopen_{start} = true\ AND\ visited(S_{start})then$
51 :     seeds = seeds $\cup S_{start}$
52:     End for
53:   Function CUT BRANCHES(s)
54:   Visited(s) = false
55:      Parent(s) = NULL
56:   REMOVE OPEN(s)
57 :     for all $s' \in Succ(s)\ do$
58 :     if visited $(s')$ AND NOT parent $(s')$ = s then
59 :         seeds = seeds $\cup s'$
60:  End for
61 :      for all $s' \in Pred(s)\ do$
62 :      if visited $(s')$ AND parent $(s')$ = s then
63 :               CUT BRANCHES $(s')$
64:   End for
65 : Output $D^*(RPP) = \{D^*(RPP_1), D^*(RPP_2), \ldots, D^*(RPP_n)\}$
```

5. Mathematical Modeling

This section presents the mathematical structures, equations, and algorithms that are central to this resource-provisioning study, providing a rigorous basis for our theoretical and empirical investigations. The performance of the D* extra lite framework is analyzed through mathematical modeling. For modeling purposes, a limited cloud setup that is composed of a predefined set of user tasks and resource center instances is considered. The performance objectives considered for evaluation purposes are power consumption (PC(D* extra lite)), resource utilization (RU(D* extra lite)), total execution time (TET(D* extra lite)), and learning rate (LR$\left(\text{D}^*\right.$ extra lite)). The expected value analysis is performed to determine the performance metrics for the future time FT. Finally, the performance of the proposed D* extra lite framework is compared with three existing works: EW1-TP [13], EW2-AC [14], and EW3-GC [16].

PO1: Power Consumption (PC(D* extra lite)): The power consumption of D* extra lite is mainly influenced by the power consumed by the virtual machines during the maximum power consumption state $PC\left(vm_i^{max}\right)$ and minimum power consumption state $PC\left(vm_i^{min}\right)$. The PC(D* extra lite) is low as D* extra lite performs smooth matching of tasks and resource instances by finding safer paths, whereas the power consumption of EW1 is comparatively higher as it performs static classification of tasks as computation-intensive, memory-intensive, and storage-intensive by considering the present traffic scenario and ignoring the history. The power consumption of EW2 and EW3 is higher than that of EW1 due to loose function approximation, poor scalability, and inappropriate selection of resources.

$$E\left(\frac{PC(D^*\ extra\ lite)}{D^*(\mathrm{TSP})},\ FT\right)=\int_x^y\frac{\sum_{a\epsilon\pi}PC(D^*\ extra\ lite)(a)}{|D^*(\mathrm{TSP})|}$$

$$E\left(\frac{PC(D^*\ extra\ lite)}{D^*(\mathrm{TSP})},\ FT\right)=\sum_{d\in D}d\int_x^y\frac{\sum_{a\epsilon\pi}PC(D^*\ extra\ lite)(a)}{|D^*(\mathrm{TSP})|}$$

$$E\left(\frac{PC(D^*\ extra\ lite)}{D^*(\mathrm{TSP})},\ FT\right)=\frac{E\left(1*\pi*\sum_{i=1}^{i=n}PC(RCI_i)\right)}{P(D^*(\mathrm{TSP}))}$$

$$=\sum_{d\in D}d\int_q^Q\left[PC\left(vm_i^{max}\right)-PC\left(vm_i^{min}\right)\right]*RU(vm_i)+PC\left(vm_i^{idle}\right)\right]$$

$$=\int_q^Q Q*dP\frac{\left[PC\left(vm_i^{max}\right)-PC\left(vm_i^{min}\right)\right]*RU(vm_i)+PC\left(vm_i^{idle}\right)\right]}{D^*(\mathrm{TSP})}$$

$$\mathrm{PC(D*\ extra\ lite)}:\ E\left(\frac{PC(D^*\ extra\ lite)}{D^*(\mathrm{TSP})},\ FT\right)\approx Low$$

$$\mathrm{PC(EW1)}:\ E\left(\frac{PC(EW1)}{D^*(\mathrm{TSP})},\ FT\right)\approx Medium$$

$$\mathrm{PC(EW2)}:\ E\left(\frac{PC(EW2)}{D^*(\mathrm{TSP})},\ FT\right)\approx High$$

$$\mathrm{PC(EW3)}:\ E\left(\frac{PC(EW3)}{D^*(\mathrm{TSP})},\ FT\right)\approx High$$

PO2: Resource Utilization (RU(D* lite)): The resource utilization of D* extra lite is mainly influenced by the over-utilization of resources of resource center instance RCI_i^{over} and under-utilization of resources of resource center instance RCI_i^{under}. The RU(D* lite) is high as it combines an incremental search and a heuristic search strategy for task mapping. The resource utilization of EW1 is low due to improper formulation of the workflow model which leads to more resource wastage. The resource utilization of EW2 is moderate as the model is not capable of dealing with a large-state-space environment and takes a longer time to converge to Nash equilibrium. The RU(EW3) is low due to convergence to suboptimal solutions and poor task-routing capability.

$$\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \int_{x}^{y} \frac{\sum_{a\epsilon\pi} RU(D^{*}\ extra\ lite)(a)}{|D^{*}(\text{TSP})|}$$
$$E\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \sum_{d\in D} d\int_{x}^{y} \frac{\sum_{a\epsilon\pi} RU(D^{*}\ extra\ lite)(a)}{|D^{*}(\text{TSP})|}$$
$$E\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \frac{E\left(1*\pi*\sum_{i=1}^{i=n} RU(RCI_i)\right)}{P(D^{*}(\text{TSP}))}$$
$$E\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \sum_{d\in D} d\int_{q}^{Q} \sum_{i=1}^{i=n} \left(RCI_i^{over} - RCI_i^{under}\right)/N_{RCI}$$
$$E\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \int_{q}^{Q} Q*dP\frac{\sum_{i=1}^{i=n}\left(RCI_i^{over} - RCI_i^{under}\right)/N_{RCI}}{D^{*}(\text{TSP})}$$
$$\text{RU(D}*\text{ extra lite)}:\ E\left(\frac{RU(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) \approx High$$
$$\text{RU(EW1)}:\ E\left(\frac{RU(EW1)}{D^{*}(\text{TSP})}, FT\right) \approx Low$$
$$\text{RU(EW2)}:\ E\left(\frac{RU(EW2)}{D^{*}(\text{TSP})}, FT\right) \approx Medium$$
$$\text{RU(EW3)}:\ E\left(\frac{RU(EW3)}{D^{*}(\text{TSP})}, FT\right) \approx Low$$

PO3: Total Execution Time (TET(D* lite)): The total execution time of D* extra lite is mainly influenced by the total execution time of the Indetermsoft set function of user tasks ISF(UT_i), Indetermsoft set function of resource instances $ISF(RCI_i)$, and D* policy $D^{*}(Tsp_i)$. The TET(D* lite) is low as it is capable of handling dynamic obstacles in cloud scenarios using an incremental search strategy. The TET(EW1) is moderate due to inappropriate mapping of resources. The TET(EW2) is very high because of poor workflow classification and softmax parametrization taking an infinite period of time for convergence. The TET(EW3) is low due to bad local search capability and a poor convergence rate.

$$E\left(\frac{TET(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \int_{x}^{y} \frac{\sum_{a\epsilon\pi} TET(D^{*}\ extra\ lite)(a)}{|D^{*}(\text{TSP})|}$$
$$E\left(\frac{TET(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) = \sum_{d\in D} d\int_{x}^{y} \frac{\sum_{a\epsilon\pi} TET(D^{*}\ extra\ lite)(a)}{|D^{*}(\text{TSP})|}$$
$$= \frac{E\left(1*\pi*\sum_{i=1}^{i=m} TET(ISF(UT_i)) + \sum_{i=1}^{i=n} TET(ISF(RCI_i)) + \sum_{i=1}^{i=n} TET(D^{*}(Tsp_i))\right)}{P(D^{*}(\text{TSP}))}$$
$$= \sum_{d\in D} d\int_{q}^{Q} \sum_{i=1}^{i=m} TET(ISF(UT_i)) + \sum_{i=1}^{i=n} TET(ISF(RCI_i)) + \sum_{i=1}^{i=n} TET(D^{*}(Tsp_i))$$
$$= \int_{q}^{Q} Qdp\frac{\sum_{i=1}^{i=m} TET(ISF(UT_i)) + \sum_{i=1}^{i=n} TET(ISF(RCI_i)) + \sum_{i=1}^{i=n} TET(D^{*}(Tsp_i))}{D^{*}(\text{TSP})}$$
$$\text{TET(D}*\text{ extra lite)}:\ E\left(\frac{TET(D^{*}\ extra\ lite)}{D^{*}(\text{TSP})}, FT\right) \approx Low$$
$$\text{TET(EW1)}:\ E\left(\frac{TET(EW1)}{D^{*}(\text{TSP})}, FT\right) \approx Medium$$
$$\text{TET(EW2)}:\ E\left(\frac{TET(EW2)}{D^{*}(\text{TSP})}, FT\right) \approx High$$
$$\text{TET(EW3)}:\ E\left(\frac{TET(EW3)}{D^{*}(\text{TSP})}, FT\right) \approx Low$$

PO4: Learning Rate (LR$\left(\text{D}^{*}\right.$ extra lite)): The learning rate of D* extra lite is mainly influenced by the total execution time for the formulation of the first task-scheduling policy $TET(D^{*}(TSP_1))$ and the number of task-scheduling policies formulated $D^{*}(TSP)$.

The $\mathrm{LR}\left(\mathrm{D}^{*}\text{ extra lite}\right)$ is very high due to high planning efficiency and provides a fast reliable path from user tasks to resource center instances. The LR(EW1) is medium as it makes poor task-scheduling decisions considering limited information about user tasks. The LR(EW2) is low as it does not consider the computation-sensitive and storage-sensitive features of the tasks. The LR(EW3) is low due to low learning accuracy and converging to a low-optimality solution as it is easily affected by external factors.

$$E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) = \int_{x}^{y} \frac{\sum_{a\epsilon\pi} LR(D^{*}\ extra\ lite)(a)}{|D^{*}(\mathrm{TSP})|}$$
$$E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) = \sum_{d\in D} d\int_{x}^{y} \frac{\sum_{a\epsilon\pi} LR(D^{*}\ extra\ lite)(a)}{|D^{*}(\mathrm{TSP})|}$$
$$E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) = \frac{E(1*\pi*TET(D^{*}(TSP_{1}))*N(D^{*}(TSP)))}{P(D^{*}(\mathrm{TSP}))}$$
$$E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) = \sum_{d\in D} d\int_{q}^{Q} TET(D^{*}(TSP_{1}))*N(D^{*}(TSP))$$
$$E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) = \int_{q}^{Q} Q*dP\frac{TET(D^{*}(TSP_{1}))*N(D^{*}(TSP))}{D^{*}(\mathrm{TSP})}$$
$$\mathrm{LR}(\mathrm{D}*\ \text{extra lite}):\ E\left(\frac{LR(D^{*}\ extra\ lite)}{D^{*}(\mathrm{TSP})}, FT\right) \approx High$$
$$\mathrm{LR}(\mathrm{EW1}):\ E\left(\frac{LR(EW1)}{D^{*}(\mathrm{TSP})}, FT\right) \approx Medium$$
$$\mathrm{LR}(\mathrm{EW2}):\ E\left(\frac{LR(EW2)}{D^{*}(\mathrm{TSP})}, FT\right) \approx Low$$
$$\mathrm{LR}(\mathrm{EW3}):\ E\left(\frac{LR(EW3)}{D^{*}(\mathrm{TSP})}, FT\right) \approx Low$$

6. Results and Discussion

This section presents our findings and interprets their significance in the context of resource-provisioning studies. As the key outcome of our research, a summary of the data and insights that have emerged from the analyses is presented. Also, a stage is set for a comprehensive comparison of our work with three recent existing works.

6.1. Experimental Setup

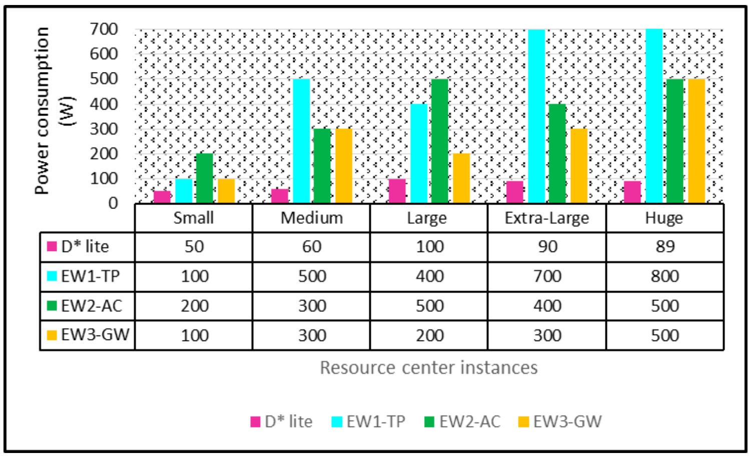
The CloudSim 3.0.3 framework is used for the modeling and simulation of the proposed Indetermsoft-set-based D* extra lite framework [18,19]. The experimental setup for execution is as follows: The resource center consists of three systems: Intel 5150 CPU containing 40 core processors with 3.0 GHz clock speed, Xeon 4150 CPU containing 45 core processors with 4.0 GHz clock speed, and Silver 6150 CPU containing 50 core processors with 2.0 GHz clock speed. Four host machine configurations are used: Host Machine 1 = {PE = 2 (small), MIPS = 2660, RAM = 4 GB, Storage = 160 GB, $PC\left(vm_{i}^{max}\right)$ = 135, $PC\left(vm_{i}^{min}\right)/PC\left(vm_{i}^{idle}\right)$ = 93.7}, Host Machine 2 = {PE = 4 (medium), MIPS = 3067, RAM = 8 GB, Storage = 250 GB, $PC\left(vm_{i}^{max}\right)$ = 113, $PC\left(vm_{i}^{min}\right)/PC\left(vm_{i}^{idle}\right)$ = 42.3}, Host Machine 3 = {PE = 12 (large), MIPS = 3067, RAM = 16 GB, Storage = 500 GB, $PC\left(vm_{i}^{max}\right)$ = 222, $PC\left(vm_{i}^{min}\right)/PC\left(vm_{i}^{idle}\right)$ = 58.4}, and Host Machine 4 = {PE = 24 (Extra-large), MIPS = 4067, RAM = 18 GB, Storage = 600 GB, $PC\left(vm_{i}^{max}\right)$ = 322, $PC\left(vm_{i}^{min}\right)/PC\left(vm_{i}^{idle}\right)$ = 68.4}. Four virtual machine configurations are used: Virtual Machine 1 = {Host Machine 1, VM type = small, PE = 1, MIPS = 500, RAM (GB) = 0.5, Storage (GB) = 40}, Virtual Machine 2 = {Host Machine 2, VM type = medium, PE = 2, MIPS = 1000, RAM (GB) = 1.0, Storage (GB) = 60}, Virtual Machine 3 = {Host Machine 3, VM type = large, PE = 3, MIPS = 1500, RAM (GB) = 2.0, Storage (GB) = 80}, and Virtual Machine 4 = {Host Machine 4, VM type = Extra-large, PE = 4, MIPS = 2000, RAM (GB) = 3.0, Storage (GB) = 100}. Two real-time datasets are used for implementation: the Google Cluster dataset and the Bitbrains dataset [20].

6.2. Google Cluster Dataset

The Google Cluster dataset is composed of physical resources such as memory, CPU, and disk storage. The user task set is composed of 672,300 tasks that are executed over 12,500 host machines for a period of 30 days. The experiments are conducted with varying sizes of resource center instances considering 100 (Tiny), 300 (Small), 700 (Large), 900 (Extra Large), and 1000 (Huge) host machines. The ratio of virtual machines to host machines is 2:1. As per the user task set resource demands, the virtual machines are allocated for every 5 min randomly. The user task set is created with a virtual machine ratio of 60 percent. The number of virtual machines varies over time, and each user task set can keep 0 to 20 virtual machines. The user task set exhibits varying resource requirements. There will be user task set demand of workload burst conditions in terms of massive failure of virtual machines, and at peak service hours, situations of overloads and resource contention will occur. Each experiment is conducted for a time period of 100 min to analyze the performance of the proposed D* extra lite framework dynamically with respect to the performance objectives mentioned.

PO1: Power Consumption (PC(D* extra lite))

A graph of varying resource center instances versus power consumption is shown in Figure 2. It is observed from the graph that the power consumption of D* extra lite is minimal over varying sizes of resource center instances. The framework is focused and performs smooth matching between tasks and resource instances by finding safer paths. The power consumption of EW1-TP is very high as it performs static classification of tasks as computation-intensive, memory-intensive, and storage-intensive by considering the present traffic scenario and ignoring the history. The power consumptions of EW2-AC and EW3-GW are moderate due to loose function approximation, poor scalability, and inappropriate selection of resources. The workflow model considered in EW2-AC and EW3-GW has limited information about the task parameters which makes it not suitable for real-world situations.

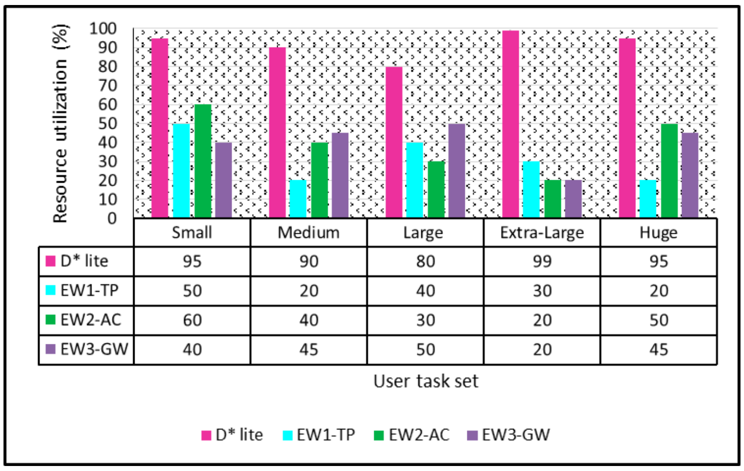


	Small	Medium	Large	Extra-Large	Huge
■ D* lite	50	60	100	90	89
■ EW1-TP	100	500	400	700	800
■ EW2-AC	200	300	500	400	500
■ EW3-GW	100	300	200	300	500

Figure 2. Resource center instances versus power consumption (W).

PO2: Resource Utilization (RU(D* lite))

A graph of varying sizes of user task sets versus resource utilization is shown in Figure 3. The resource utilization of D* extra lite is optimal over all sizes of user tasks as it combines incremental and heuristic search strategies for task mapping. The resource utilization of EW1-TP is poor due to improper formulation of the workflow model which does not suit the real-time scenario of cloud systems. The structural features extracted from the tasks are poor, which leads to resource wastage. The resource utilization of EW2-AC and EW3-GW is moderate due to convergence to suboptimal solutions and poor task-routing capability. They are not capable of dealing with a large-state-space environment, and the resource allocation models take longer time to converge to Nash equilibrium.

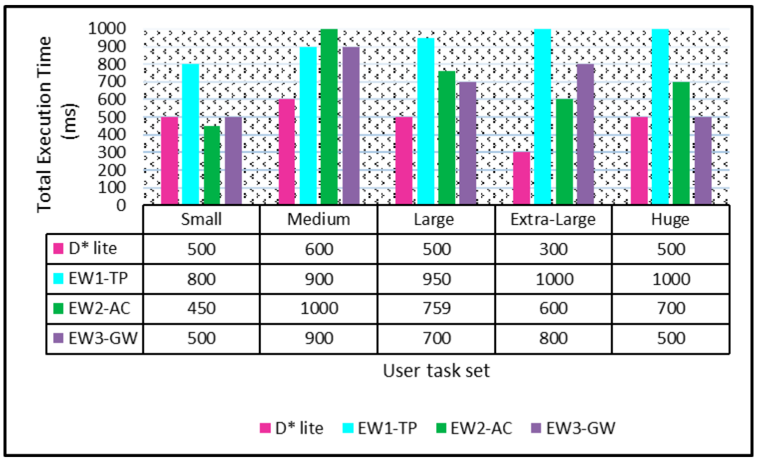


	Small	Medium	Large	Extra-Large	Huge
■ D* lite	95	90	80	99	95
■ EW1-TP	50	20	40	30	20
■ EW2-AC	60	40	30	20	50
■ EW3-GW	40	45	50	20	45

Figure 3. User task set versus resource utilization.

PO3: Total Execution Time (TET(D* lite))

A graph of varying sizes of user tasks set versus total execution time (ms) is shown in Figure 4. The total execution time of D* extra lite is considerably less for all sizes of user task sets due to effective task scheduling despite dynamic obstacles in cloud scenarios. The total execution time of EW1-TP and EW2-AC is very high due to the poor classification of workflow and inappropriate mapping of resources, which cause the execution time to spike. The softmax parametrization also takes an infinite period of time for convergence to promising solutions. The total execution time of EW3-GW is moderate due to poor local search capability and poor convergence rate. Also, the resource management model provides minimal extensibility of a service portfolio.

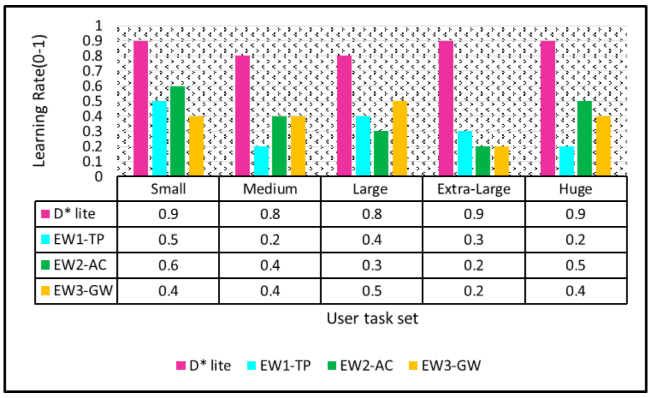


	Small	Medium	Large	Extra-Large	Huge
■ D* lite	500	600	500	300	500
■ EW1-TP	800	900	950	1000	1000
■ EW2-AC	450	1000	759	600	700
■ EW3-GW	500	900	700	800	500

Figure 4. User task set versus total execution time (ms).

PO4: Learning Rate (LR$\left(D^* \text{ lite}\right)$)

A graph of varying sizes of user task sets versus learning rate is shown in Figure 5. The learning rate of D* extra lite is very high due to high planning efficiency and provides a fast reliable path from user tasks to resource center instances. The learning rate of EW1-TP is very low as it makes poor task-scheduling decisions considering limited information on user tasks. It does not consider the computation-sensitive and storage-sensitive features of the tasks. The learning rates of EW2-AC and EW3-GW are moderate due to low learning accuracy and convergence to low-optimality solutions. They also fail to capture the complexity of real-world situations, and the outcomes of the approach are easily affected by external factors.



	Small	Medium	Large	Extra-Large	Huge
D* lite	0.9	0.8	0.8	0.9	0.9
EW1-TP	0.5	0.2	0.4	0.3	0.2
EW2-AC	0.6	0.4	0.3	0.2	0.5
EW3-GW	0.4	0.4	0.5	0.2	0.4

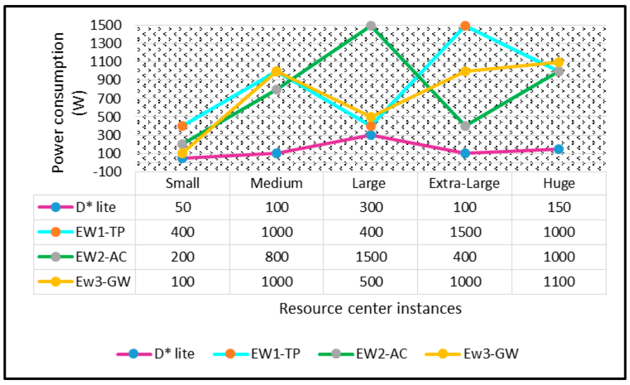
Figure 5. User task set versus learning rate.

6.3. Bitbrains Dataset

The Bitbrains dataset is composed of performance metrics related to rapid data storage of 2830 virtual machines over distributed resource center instances for a period of one month. It mainly constitutes information related to the percentage of CPU usage, memory consumed (kilobytes), memory sanctioned (kilobytes), network capacity, throughput, etc. The ratio of virtual machines to host machines is 4:2, and per the user task set resource demand, the virtual machines are allocated for every minute randomly. The user task sets are created with a virtual machine ratio of 40 percent. The number of virtual machines varies over time, and each user task set can keep 0 to 10 virtual machines. The experiments are conducted for different resource center instances for 24 h over a period of 60 days. Resource center instances are composed of 200 (Tiny), 400 (Small), 600 (Large), 800 (Extra-large), and 1000 (Huge) host machines. The user task sets exhibit varying resource requirements and often lead to burst conditions at peak service hours. Each experiment is executed for a time period of 600 min to analyze the performance of the proposed D* extra lite framework dynamically with respect to the performance objectives mentioned.

PO1: Power Consumption (PC(D* lite))

A graph of varying sizes of resource center instances versus power consumption is shown in Figure 6. It is observed from the graph that the power consumption of D* extra lite is considerably less over varying sizes of resource center instances from small to huge. It easily navigates the tasks to the resource center instances in a highly dynamic environment by using a priority queue to minimize the effect of reordering. The power consumption of EW3-GW is moderate because of poor exploration capability when exposed to a large state space. The convergence speed is never satisfied due to insufficient handling of task diversities. The power consumption of EW1-TP and EW2-AC is very high since it takes a long time (infinite time) to converge to Nash equilibrium under linear function approximation. Also, the tendency towards the selection of inappropriate solutions is high, and the ability to handle enormous computational resources for larger task sets is poor.

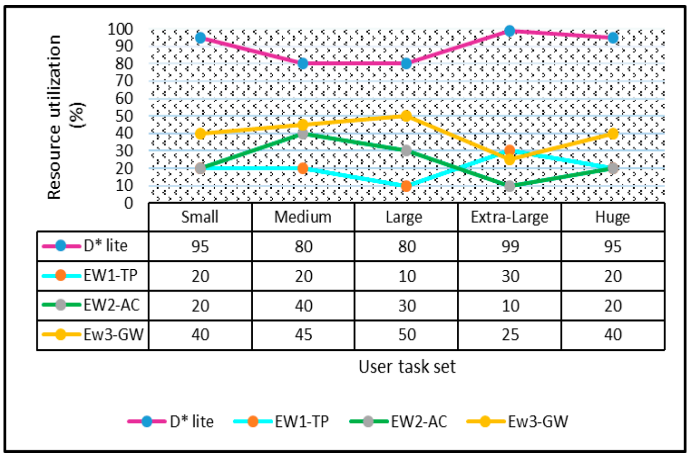


	Small	Medium	Large	Extra-Large	Huge
D* lite	50	100	300	100	150
EW1-TP	400	1000	400	1500	1000
EW2-AC	200	800	1500	400	1000
Ew3-GW	100	1000	500	1000	1100

Figure 6. Resource center instances versus power consumption (W).

PO2: Resource Utilization (RU(D* lite))

A graph of varying sizes of user task sets versus resource utilization is shown in Figure 7. The resource utilization rate of D* extra lite is very high for varying sizes of user task sets. It simplifies the maintenance of user task set priority by efficiently analyzing the program workflow. The resource utilization of RE3-GW is average as the algorithm converges to a suboptimal solution due to an imbalance in grey wolf behavior. The resource utilization of EW1-TP and EW2-AC is very low as the task-scheduling policy gradient quality is low due to a mismatch between the critical value function and actor policy. Also, the softmax parametrization of the user task set leads to an infinite time required for convergence to a promising solution. It is also not able to handle a spike in user requests which causes resource contention in the multi-tenancy nature of a serverless platform.

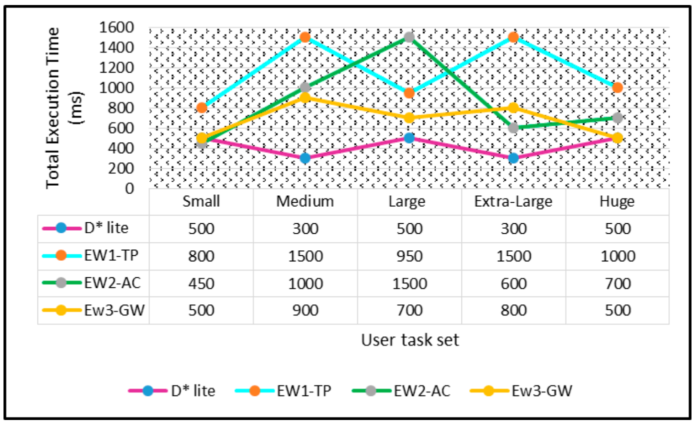


	Small	Medium	Large	Extra-Large	Huge
D* lite	95	80	80	99	95
EW1-TP	20	20	10	30	20
EW2-AC	20	40	30	10	20
Ew3-GW	40	45	50	25	40

Figure 7. User task set versus resource utilization.

PO3: Total Execution Time (TET(**D* lite**))

A graph of varying sizes of resource center instances versus total execution time is shown in Figure 8. The total execution time of D* extra lite is low for all sizes of user tasks as it is able to find the optimal path between the starting point (user test set) and ending point (resource instances) in a dynamic cloud environment. The total execution time of EW3-GW is moderate as it generates an approximate solution whose correctness of operation is not guaranteed. Service management and operating system deployment are improper due to poor approximation towards a promising solution. The total execution time of EW1-TP and EW2-AC is very high due to their limited capability to perform global search operations and susceptibility to convergence towards local suboptimal solutions. Also, the task-scheduling decisions suffer from severe scalability issues and inappropriate convergence towards promising solutions.

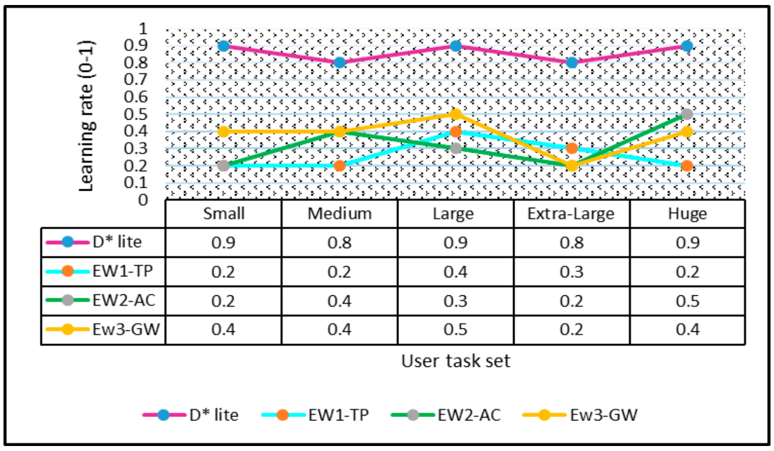


	Small	Medium	Large	Extra-Large	Huge
D* lite	500	300	500	300	500
EW1-TP	800	1500	950	1500	1000
EW2-AC	450	1000	1500	600	700
Ew3-GW	500	900	700	800	500

Figure 8. Resource center instances versus total execution time (ms).

PO4: Learning Rate ($\mathbf{LR}\left(\mathbf{D}^{*}\ \textbf{lite}\right)$)

A graph of varying sizes of resource center instances versus learning rate is shown in Figure 9. The learning rate of D* extra lite is very high for all varying sizes of user task sets since it efficiently handles uncertainty in the environment by precisely taking actions using the knowledge gained from previous searches. The learning rate of EW2-AC is very low due to neural network function approximation. Even the asymptotic behavior of the differential game approach decreases its learning rate towards promising solutions. The learning rates of EW1-TP and EW3-GW are moderate due to an imbalanced relationship between exploration and exploitation. They also fail to handle larger user requests, and the task data cannot be handled effectively with increasing user demands.



	Small	Medium	Large	Extra-Large	Huge
D* lite	0.9	0.8	0.9	0.8	0.9
EW1-TP	0.2	0.2	0.4	0.3	0.2
EW2-AC	0.2	0.4	0.3	0.2	0.5
Ew3-GW	0.4	0.4	0.5	0.2	0.4

Figure 9. Resource center instances versus learning rate.

7. Conclusions

This paper presented a novel Indetermsoft-set-based D* Extra Lite framework for resource provisioning in the cloud. The experimental evaluation was carried out using the CloudSim simulator, with the Google Cluster dataset and Bitbrains dataset. The results obtained were found to outperform three of the existing works with respect to the performance objectives of power consumption, resource utilization, total execution time, and learning rate. However, the proposed framework also suffered from limitations in terms of limited handling of highly dynamic obstacles, extensive memory usage, and increased computational overhead. As future work, complete analytical modeling and exhaustive testing of the framework are planned to address higher-end performance objectives such as fault tolerance, correctness, confidentiality, reliability, and transparency.

Author Contributions: Conceptualization, methodology, validation, and formal analysis: B.K. and S.G.S.; writing—original draft preparation: B.K.; review and editing: S.G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abid, A.; Manzoor, M.F.; Farooq, M.S.; Farooq, U.; Hussain, M. Challenges and Issues of Resource Allocation Techniques in Cloud Computing. *KSII Trans. Internet Inf. Syst.* **2020**, *14*, 2815–2839.
2. Laha, S.R.; Parhi, M.; Pattnaik, S.; Pattanayak, B.K.; Patnaik, S. Issues, Challenges and Techniques for Resource Provisioning in Computing Environment. In Proceedings of the 2020 2nd International Conference on Applied Machine Learning (ICAML), Changsha, China, 16–18 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 157–161.

3. Maenhaut, P.J.; Volckaert, B.; Ongenae, V.; De Turck, F. Resource management in a containerized cloud: Status and challenges. *J. Netw. Syst. Manag.* **2020**, *28*, 197–246. [CrossRef]
4. Kabir, H.D.; Khosravi, A.; Mondal, S.K.; Rahman, M.; Nahavandi, S.; Buyya, R. Uncertainty-aware decisions in cloud computing: Foundations and future directions. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–30. [CrossRef]
5. Li, B.; Tan, Z.; Arreola-Risa, A.; Huang, Y. On the improvement of uncertain cloud service capacity. *Int. J. Prod. Econ.* **2023**, *258*, 108779. [CrossRef]
6. Smarandache, F. *Introduction to the IndetermSoft Set and IndetermHyperSoft Set*; Infinite Study; University of New Mexico: Albuquerque, NM, USA, 2022; Volume 1.
7. Smarandache, F. *New Types of Soft Sets: HyperSoft Set, IndetermSoft Set, IndetermHyperSoft Set, and TreeSoft Set*; Infinite Study; American Scientific Publishing Group: Gretna, LA, USA, 2023.
8. Smarandache, F.; Abdel-Basset, M.; Broumi, S. (Eds.) *Neutrosophic Systems with Applications (NSWA)*; Infinite Study; Sciences Force LLC: Clifton, NJ, USA, 2023; Volume 3.
9. Xie, K.; Qiang, J.; Yang, H. Research and optimization of d-start lite algorithm in track planning. *IEEE Access* **2020**, *8*, 161920–161928. [CrossRef]
10. Ren, Z.; Rathinam, S.; Likhachev, M.; Choset, H. Multi-objective path-based D* lite. *IEEE Robot. Autom. Lett.* **2022**, *7*, 3318–3325. [CrossRef]
11. Tuli, S.; Gill, S.S.; Xu, M.; Garraghan, P.; Bahsoon, R.; Dustdar, S.; Sakellariou, R.; Rana, O.; Buyya, R.; Casale, G.; et al. HUNTER: AI based holistic resource management for sustainable cloud computing. *J. Syst. Softw.* **2022**, *184*, 111124. [CrossRef]
12. Chouliaras, S.; Sotiriadis, S. An adaptive auto-scaling framework for cloud resource provisioning. *Future Gener. Comput. Syst.* **2023**, *148*, 173–183. [CrossRef]
13. Kumar, M.S.; Choudhary, A.; Gupta, I.; Jana, P.K. An efficient resource provisioning algorithm for workflow execution in cloud platform. *Clust. Comput.* **2022**, *25*, 4233–4255. [CrossRef]
14. Mao, W.; Qiu, H.; Wang, C.; Franke, H.; Kalbarczyk, Z.; Iyer, R.; Basar, T. A mean-field game approach to cloud resource management with function approximation. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 36243–36258.14.
15. Schuler, L.; Jamil, S.; Kühl, N. AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 804–811.
16. Sangeetha, S.B.; Sabitha, R.; Dhiyanesh, B.; Kiruthiga, G.; Yuvaraj, N.; Raja, R.A. Resource management framework using deep neural networks in multi-cloud environment. In *Operationalizing Multi-Cloud Environments: Technologies, Tools and Use Cases*; Springer: Cham, Switzerland, 2022; pp. 89–104.
17. Saxena, D.; Gupta, I.; Singh, A.K.; Lee, C.N. A fault tolerant elastic resource management framework toward high availability of cloud services. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 3048–3061. [CrossRef]
18. Habaebi, M.H.; Merrad, Y.; Islam, M.R.; Elsheikh, E.A.; Sliman, F.M.; Mesri, M. Extending CloudSim to simulate sensor networks. *Simulation* **2023**, *99*, 3–22. [CrossRef]
19. Barbierato, E.; Gribaudo, M.; Iacono, M.; Jakobik, A. Exploiting CloudSim in a multiformalism modeling approach for cloud based systems. *Simul. Model. Pract. Theory* **2019**, *93*, 133–147. [CrossRef]
20. Shen, S.; van Beek, V.; Iosup, A. Statistical characterization of business-critical workloads hosted in cloud datacenters. In Proceedings of the International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4–7 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 465–474.